

CRITERION A: PLANNING

DEFINING THE PROBLEM

Connect Four is a popular game designed by Howard Wexler in 1974. The players take turns to drop in their counters and the goal of the game is to connect four of one's own colour counters in a row horizontally, vertically or diagonally on the board. The aim of my project is to create a computerised version of the game.



(Appendix 1 shows the simplified rules of the game).

Having identified the problem, my computer science teacher agreed to be my supervisor (advisor) and my friend offered to be a tester and end-user (client) as he enjoys playing Connect Four.

RATIONALE FOR SOLUTION

The reason for computerising the traditional game is to understand how the opponent plays and create strategies. A computerised version of the game can allow players to practice and develop their strategies by playing against the artificial intelligence to understand winning combinations and the AI also allows you to play the game without a partner so you can practice or play for entertainment with the computer. Finally, setting up the game after every win is tedious and takes too long so a computerised version is quicker and easier.

Having researched different programming languages, I decided to create my solution in Visual Studio 2010 C# because it has the following features:

- Graphical user interface – allows the client to visually see the board and easily drop counters in by a click of a button (creates a user-friendly interface).
- Objects can be created with changing visual characteristics – allows me, as a designer, to change the colour of the spaces when different coloured counters are dropped in and allows the client to understand the board.
- The program has an installed debugger that makes programming easier as I can step into my code and work through it to find an error.
- C# is an object-oriented programming language – this allows me to use inheritance, encapsulation and polymorphism to make programming efficient and minimise duplicated coding.
- C# is free to download and is multi-platform.

SUCCESS CRITERIA

Having talked to my client, I decided that to be a successful solution, my system will:

- Make use of a graphical user interface to show a graphical representation of the board – *this will be designed to follow the client's preferred colour scheme.*
- Identify the position where the user wishes to place their counter and replace this position with the counter if the move is valid – *the client has stated that he would like the game to keep the traditional colours of the counters, which will be taken into consideration when designing as it is a user requirement*
- Correctly identify a situation where a player has won and automatically stop play or identify when the board is full so the game is a draw – *the winner should be declared in the middle of the screen and draws should be less frequent in a one player game, as told by the client.*
- Use algorithmic thinking to identify a situation where the computer AI has won – *this is to develop an efficient one player game so the client can play by himself and develop strategies, which is a user requirement*
- Allow the user to restart a new game – *this small feature would benefit the client as it makes the game more user-friendly.*

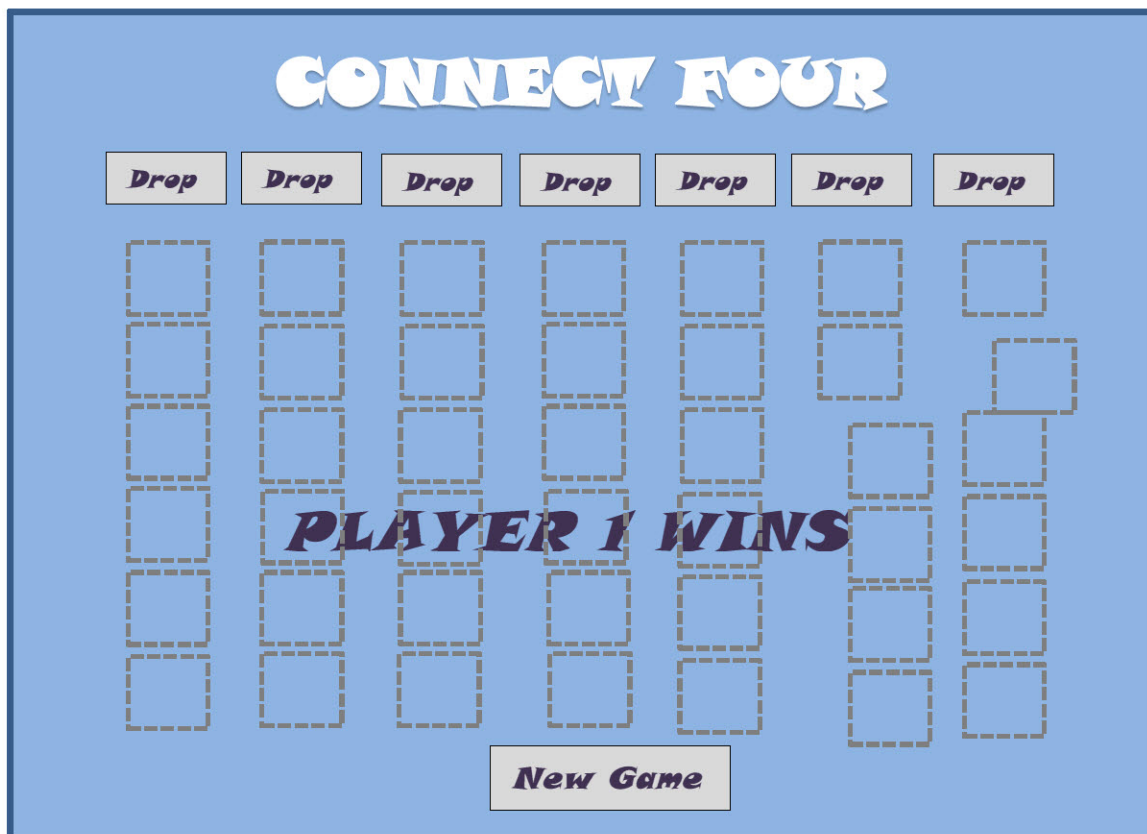
CRITERION B: DESIGN

OBJECT DESIGN

The program will have two forms. The first will consist of the title and two buttons to give the choice of picking to play a one player or a two player game. I have chosen to use a blue, purple and white colour scheme as they are the client's preferred colours.



The graphical user interface for the second form is based on the traditional Connect Four board, with 7 columns and 6 rows. The screen will include the title at the top and 7 buttons to choose the column to drop the counter in. In case of a win, a label will become visible to declare the winner and a button to start a new game.



The design for the players' counters that will be dropped into the board use two different colours so the players can be differentiated. The colours are similar to the traditional games colours but they are square, rather than circles to make the interface design simpler and easier to develop.

Player 1

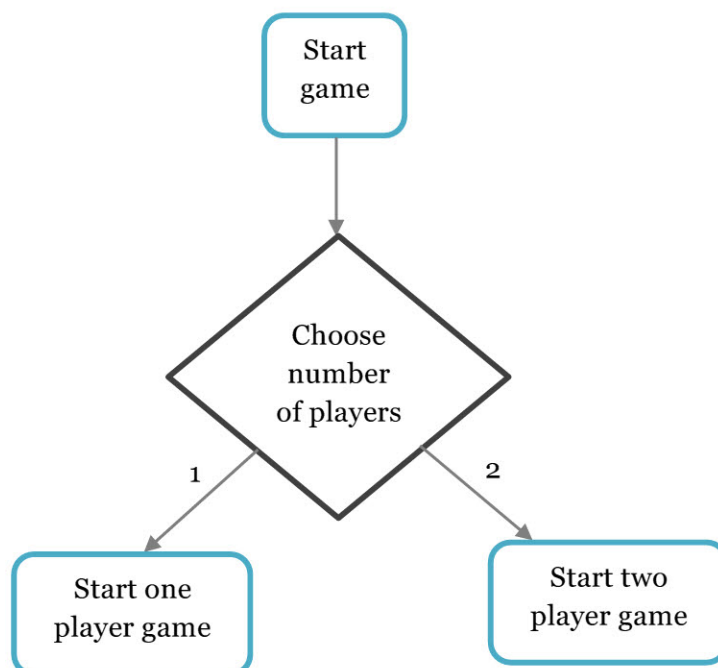


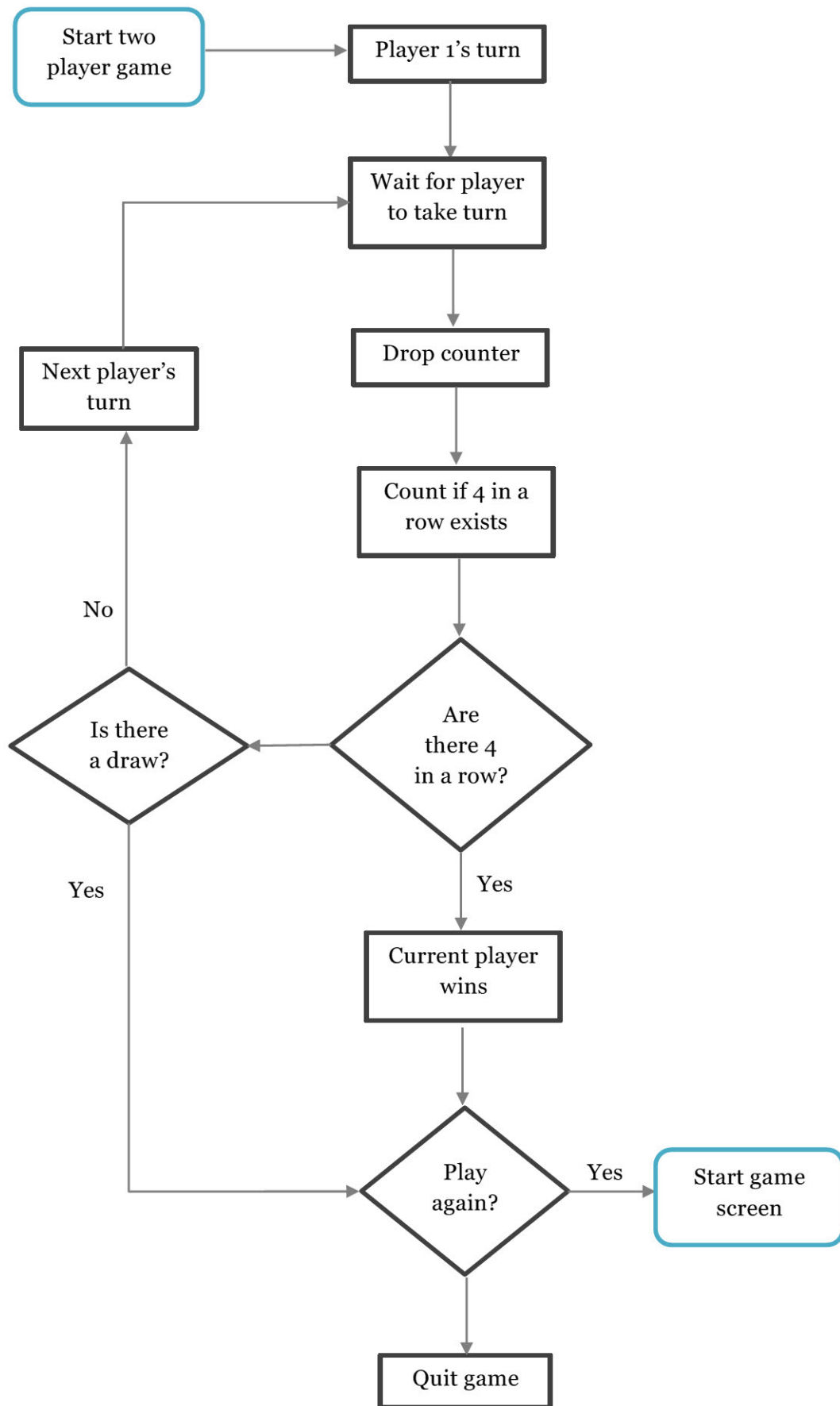
Player 2

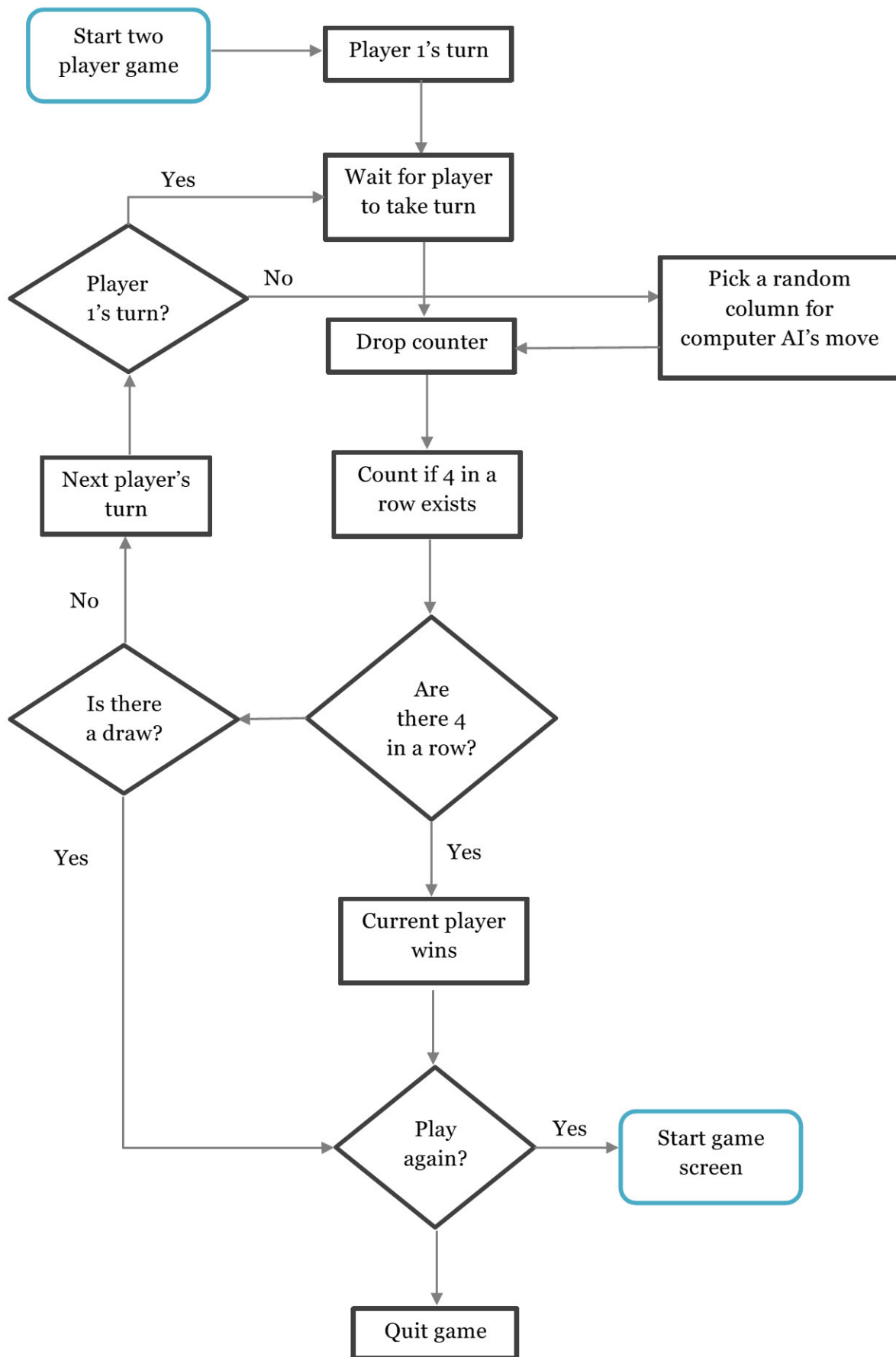


PROCESSING FLOWCHART

The flowcharts outline the overall logic of the program:

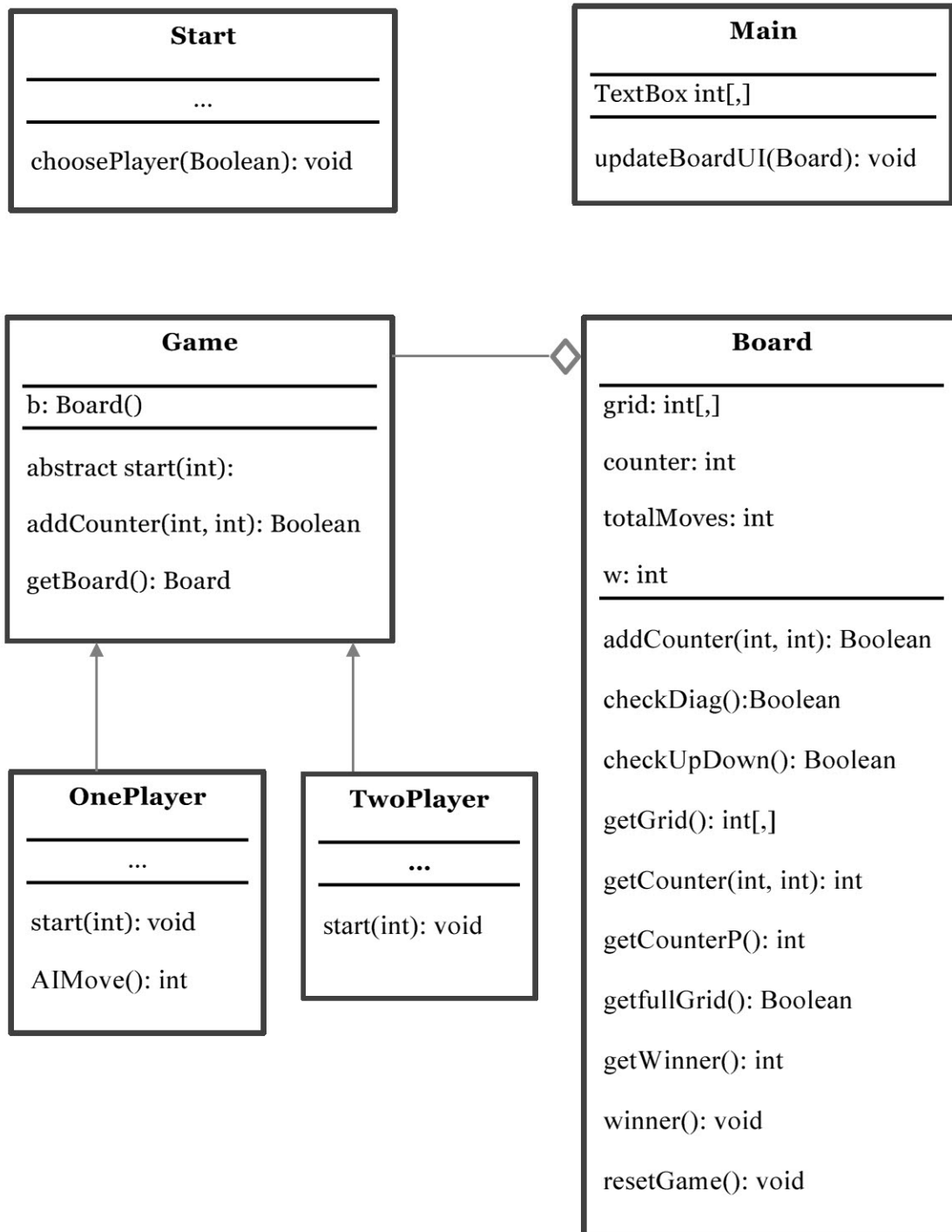






CLASS DIAGRAM

I will code the program using object-oriented programming so it will consist of 6 classes, which make use of inheritance, encapsulation and polymorphism. This class diagram shows the basic structure of the classes.



DEVELOPMENT PSUEDOCODE

These methods are written in psuedocode to understand the basic function of the main methods that are needed in the program.

Choose one/two player:

```
Boolean TWOPLAYER //parameter variable indicating the button clicked  
if TWOPLAYER= true  
    then new two player game;  
else if TWOPLAYER = false  
    then new one player game;  
end else if  
show form 2  
hide form 1
```

Start (two player):

```
int COL //parameter variable showing the column that the user wants  
add counter to COL  
if win = true  
    then state winner and reset game  
end if
```

Start (one player):

```
int COUNTER = 0 //to keep track of which player's turn it is
int COL //parameter variable showing the column that the user wants
if COUNTER % 2 = 0
    then add counter to COL
    then add counter to calculated best AI position
end if
COUNTER = COUNTER + 1
if win = true
    then state winner and reset game
end if
```

Calculate AI move:

```
Random RND // creates an instance of a random number
int POS = RND(0 to 6) //assigns position a random number from 0 to 6
return POS
```

Add counter:

```
Boolean ADDC = false //indicates if counter has been added successfully
int X = 0 // tracks the filled positions in a column
int COUNTER //parameter variable to differentiate between players (1 or 2)
int COLUMN //parameter variable to show the column to drop counter in
if counter % 2 = 0
    then COUNTER = 1
else if counter % 2 = 1
    then COUNTER = 2
end else if
loop while X < 6 AND grid[X, COLUMN] ≠ empty
    X = X + 1
end loop
if X < 6
    grid[X, COLUMN] = COUNTER
    ADDC = true
    COUNTER = COUNTER + 1
    TOTALMOVES = TOTALMOVES + 1
else ADDC = false
end else
return ADDC
```


Check for horizontal and vertical win:

```
int COUNT = 0 //used to compare the number of identical counters in a row
int PLAYER = 0 //keeps track of which player has dropped a counter
Boolean WIN = false //indicates if either player has won
if counter % 2 = 0
    then PLAYER = 1
else if counter % 2 = 1
    then PLAYER = 2
end else if
loop ROW from 0 to 5
    loop COL from 0 to 6
        if GRID[ROW, COL] = PLAYER
            COUNT = COUNT + 1
        else COUNT = 0
        end else if
        if COUNT = 4
            then WIN = true
        end if
    end loop
end loop
loop COL from 0 to 6
    loop ROW from 0 to 5
        if GRID[ROW, COL] = PLAYER
            COUNT = COUNT + 1
        else COUNT = 0
        end else if
        if COUNT = 4
            then WIN = true
        end if
    end loop
end loop
return WIN
```

Check for diagonal win:

```
int COUNT = 0 //used to compare the number of identical counters in a row
int PLAYER = 0 //keeps track of which player has dropped a counter
Boolean WIN= false //indicates if either player has won
int OFFSET //keeps track of the position that is being checked for a win
if counter % 2 = 0
    then PLAYER = 1
else if counter % 2 = 1
    then PLAYER = 2
end else if
loop ROWSTARTINDEX from 0 to 2
    loop COLSTARTINDEX fom 0 to 3
        COUNT = 0
        OFFSET = 0
        loop COL from COLSTARTINDEX
            if GRID[ROWSTARTINDEX + OFFSET, COL] = PLAYER
                then COUNT = COUNT + 1
            else COUNT = 0
            end else if
            if COUNT = 4
                then WIN = true
            end if
        end loop
    end loop
end loop
loop ROWSTARTINDEX from 0 to 2
    loop COLSTARTINDEX from 6 to 1
        COUNT = 0
        OFFSET = 0
        loop COL from COLSTARTINDEX
            if GRID[ROWSTARTINDEX + OFFSET, COL] = PLAYER
                then COUNT = COUNT + 1
            else COUNT = 0
            end else if
            if COUNT = 4
                then WIN = true
            end if
        end loop
    end loop
end loop
```

Update graphical user interface:

```
loop ROW from 0 to 5
  loop COL from 0 to 6
    if counter in position GRID [ROW, COL] = 1
      then set back colour = red
    else if counter in position GRID[ROW, COL] = 3
      then set back colour = yellow
    end else if
  end loop
end loop
```

Reset game:

```
show form 1
hide form 2
```

TEST PLAN

These tests need to be carried out to ensure that the program fulfils the success criteria and user requirements.

ACTION TO TEST	METHOD OF TESTING/EXPECTED RESULT
Second screen is displayed with correct game type	Play multiple times to ensure that the game shown is for the desired number of players
Player changes after each move	Try waiting and seeing if the computer takes a second turn in succession. Try to go when no player turn in a one-player game.
Clicking on button allows the user to drop their counter into that column	Check the array holding the numbers corresponding to the positions on the board and check visually for the correct coloured counter on the graphical board.
A turn is invalid if a player attempts to drop a counter into a full column	Try to add a 7 th counter to a full column and check if the player is given another attempt at the turn as the move is considered invalid.
A win is found when there are 4 counters of one colour in the horizontal, vertical or diagonal direction	Try different combinations of winning moves in all directions to check if the game spots the winning move.
A game is declared a draw if the board is filled with no winner.	Try to fill the entire board with neither player winning. The draw should be declared and the option to start a new game should become visible.
How effective is the algorithm for calculating the computer's move using AI?	Can be seen from multiple plays and finding how often the computer is able to win against the human opponent.
The game ends when there is a draw or either one of the two players or the computer wins	Check the counters visually to find the point in time where there is a win or draw and the game should end automatically.
Correct winner allocated and declared	Check the winner visually and see for each game individually.
The button to create a new game redirects the player back to the first screen	Play multiple times to make sure that the button redirects to the correct screen and changes visibility.

CRITERION B: RECORD OF TASKS

TASK NO.	PLANNED ACTION	PLANNED OUTCOME	TIME ESTIMATED	TARGET COMPLETION DATE	CRITERION
1	Decide on project	Decide on a project that has a suitable level of complexity that my CS teacher agrees on.	1 week	20/06/13	A
2	Discussion with CS teacher on proposal	My teacher approves the proposal and recommends some functions.	3 days	23/06/13	A
3	Research programming languages and functions	Research and understand the functions that would be needed for my code that I don't know. Also, decide on a programming language to use.	1 week	30/06/13	A
4	Draw up schedule	Create a schedule with basic dates and tasks to be done.	2 days	02/07/13	A
5	Define criteria for success	Record requirements that the solution should fulfil at the end, using the client's input.	10 days	12/07/13	A
6	Complete interface design	Use the GUI design in C# to create the different forms and main board	2 weeks	26/07/13	B
7	Complete algorithmic design	Highlight the basic rules of each class and the main methods (add counter, check win).	10 days	05/08/13	B
8	Start to code objects	Code the main methods and AI, linking it to the GUI.	3 months	05/11/13	B, C
9	Product developed	Complete coding the program.	3 days	08/11/13	C
10	Solution tested by client	Client tests the final product and gives feedback for improvement.	1 week	15/11/13	C, D
11	Solution improved	Complete the improvements proposed by the client, where possible.	2 weeks	30/11/13	C, D
13	Solution given to CS advisor	Improved final solution given to advisor for testing and feedback as an expert.	1 week	07/12/13	D
14	Feedback from teacher	Receive feedback on the strengths and weaknesses of the solution.	3 days	11/12/13	D, E
15	Ideas for further improvements and extensions	Use weaknesses of the solution to think of other possible improvements that would make the coding more efficient or complex.	2 weeks	18/12/13	E
16	Finish write-up for solution	Complete the evaluation of the write-up and hand in as final copy to CS teacher.	3 weeks	08/01/14	All

CRITERION C: DEVELOPMENT

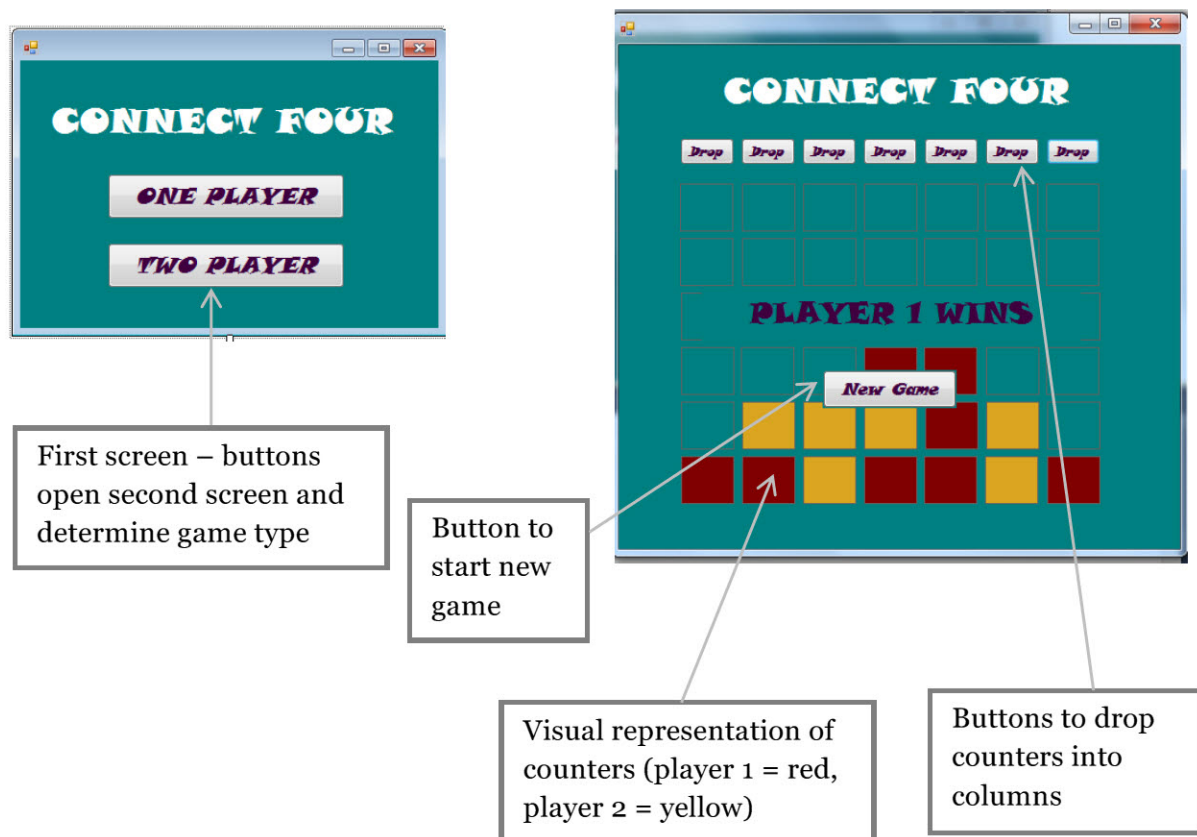
LIST OF TECHNIQUES

The techniques used in the programming include:

- Graphical user interface
- Global and local variables
- Methods
- Algorithmic thinking for AI
- Encapsulation, using get and set methods
- Inheritance of classes
- Polymorphism, using abstract methods

GRAPHICAL USER INTERFACE

This is the graphical interface of the board and the starting screen, which shows the user a visual representation of the game. It follows the client's preferred colour scheme.



This is the public method, `updateBoardUI`, which takes the board as a parameter variable to update the graphical user interface. This method is independent of the game class and position chosen so can easily be extended to update the GUI of a board of another game (it can be used for updating a universal board).

```
public void updateBoardUI(Board b){  
  
    for (int x = 0; x < 6; x++){  
    {  
        for (int z = 0; z < 7; z++){  
        {  
            if(b.getCounter(x,z) == 1){  
                boardUI[x,z].BackColor = Color.Maroon;  
            }else if(b.getCounter(x,z) == 2){  
                boardUI[x,z].BackColor = Color.Goldenrod;  
            }  
        }  
    }  
  
    if (b.getWinner() == 1)  
    {  
        textBox43.Visible = true;  
        textBox43.Text = "PLAYER 1 WINS";  
        button8.Visible = true;  
    }  
    else if (b.getWinner() == 2)  
    {  
        textBox43.Visible = true;  
        if (players == 1)  
        {  
            textBox43.Text = "COMPUTER WINS";  
        }  
        else  
        {  
            textBox43.Text = "PLAYER 2 WINS";  
        }  
        button8.Visible = true;  
    }  
    else if (b.getFullGrid() == true)  
    {  
        textBox43.Visible = true;  
        textBox43.Text = "DRAW";  
        button8.Visible = true;  
    }  
}
```

VARIABLES

Global variables are used in the three main classes of the program. The public variables used in the Main class are accessible by all classes, whilst the private variables can only be changed by using access methods – this is encapsulation.

```
Game g;  
    (to create a new instance of Game called g)  
TextBox[,] boardUI = new TextBox[6,7];  
    (two-dimensional array to hold positions of grid and update GUI)  
int players = 0;  
    (to determine if game is one or two player)  
private Board b;  
    (to create a new instance of a Board called b)  
private int[,] grid = new int[6, 7];  
    (two-dimensional array to hold positions of counters on grid)  
private int counter = 0;  
    (to keep track of no. of moves and use to calculate player turn)  
private int totalMoves = 0;  
    (to keep total no. of moves and determine if grid is full)  
private int w = 0;  
    (to hold value corresponding to winning player)
```

I used local variables in the classes and methods to keep track of variables and to use as a counter. However, the most important local variables that were defined and used in more than one instance are:

```
Boolean addC = false;  
    (to show if move was valid and counter has been added to board)  
int count = 0;  
    (to count the no. of counters in a row and check for win)  
int player = 0;  
    (to hold value of which players turn it is)  
Boolean win = false;  
    (to show if there is a win)  
int pos = rnd.Next(0, 6);  
    (to generate random position for computer's move)
```

The use of global and local variables allowed me to isolate the global variables that were needed at more than one time and these could be used in calculations and methods by assigning their values to local variables.

METHODS

The main methods used in the program are the same ones mentioned in the design section. Having written the pseudocode for these methods, I had a good understanding of how to develop and code them.

The private method choosePlayer is used to pick a one player or a two player game and is very straightforward. The method takes the user input from the button as a parameter variable to create a game with the correct no. of players, shows the second screen and hides the first. This was a simple method to code and took very little time.

```
private void choosePlayer(Boolean twoPlayer)
{
    Main main = new Main(twoPlayer);
    main.Show();
    this.Visible = false;
}
```

The private resetGame method was also very simple to code and it is linked to the new game button. When clicked, the method runs and creates a new instance of the first Start class, shows the previously hidden first screen and makes the second screen invisible.

```
private void button8_Click(object sender, EventArgs e)
{
    Start start = new Start();
    start.Visible = true;
    this.Visible = false;
}
```

The public addCounter method takes the counter of Player 1 or 2 and the column that they want to drop it in as parameter variables. If the move is valid, the counter is added to the board and the corresponding position in the two-dimensional array grid is assigned the value of the counter. Having previously coded noughts and crosses, I found this method quite simple as only the size of the board was different but the logic was very similar.

```

public Boolean addCounter(int counter, int column)
{
    Boolean addC = false;
    int x=0;

    if (this.counter % 2 == 0)
    {
        counter = 1;
    }
    else
    {
        counter = 2;
    }

    while (x<6 && grid[x,column] != 0)
    {
        x = x + 1;
    }

    if (x < 6)
    {
        grid[x, column] = counter;
        addC = true;

        this.counter = this.counter + 1;
        this.totalMoves++;
    }
    else
    {
        addC = false;
    }
    return addC;
}

```

I decided to split the code to check for a win into two different methods to make it easier to understand and manage. The public checkUpDown method checks for a win in the basic horizontal and vertical directions. It loops through the board in a constant direction (it first loops through the rows and then columns) and counts the number of counters that the current player has in a row. If the number is equal to 4, win is set to true and returned. I initially made small errors with the limits of the loop but I debugged the code to find these and rectify these errors. The program checks for a win after every counter is added but, to make it more efficient, I could program it to start checking after each player has made a minimum of 4 moves.

```

public Boolean checkUpDown()
{
    int count = 0;
    int player = 0;
    Boolean win = false;

    if (counter % 2 == 0)
    {
        player = 1;
    }
    else
    {
        player = 2;
    }

    for (int row = 0; row < 6; row++)
    {
        for (int col = 0; col < 7 ; col++){

            if (grid[row, col] == player)
            {
                count++;
            }
            else
            {
                count = 0;
            }

            if(count==4) win = true;
        }
    }

    for (int col = 0; col < 7; col++)
    {
        for (int row = 0; row < 6 ; row++){

            if (grid[row, col] == player)
            {
                count++;
            }
            else
            {
                count = 0;
            }

            if(count==4) win = true;
        }
    }

    return win;
}

```

The public checkDiag method uses more complex algorithmic thinking as it checks for a win diagonally. Similar to the previous method, this method counts the number of counters that a player has in a row in a diagonal direction and assigns win to true if count reaches 4. To make the method more efficient, I changed the limits of the loops because there are only a limited number of diagonal lines with 4 spaces so the entire board does not need to be checked. I initially found it very difficult to understand how the value of the row should be kept independently of the looping positions and my CS teacher suggested that I could use a separate local variable, offset, which is assigned the initial value of the row/column and is used to increase the value in the loop and calculate the correct position in the grid.


```

public Boolean checkDiag()
{
    int count = 0;
    int player = 0;
    Boolean win = false;
    int offset = 0;
    if (counter % 2 == 0)
    {
        player = 1;
    }
    else
    {
        player = 2;
    }
    for (int rowStartIndex = 0; rowStartIndex < 3; rowStartIndex++)
    {
        for (int columnStartIndex = 0; columnStartIndex < 4;
columnStartIndex++)
        {
            count = 0;
            offset = 0;
            for (int col = columnStartIndex; col < columnStartIndex + 4; col++)
            {
                if (grid[rowStartIndex + offset, col] == player)
                {
                    count++;
                }
                else
                {
                    count = 0;
                }
                offset++;

                if (count == 4) win = true;
            }
        }
    }
    for (int rowStartIndex = 0; rowStartIndex < 3; rowStartIndex++)
    {
        for (int columnStartIndex = 6; columnStartIndex > 2; columnStartIndex--)
        {
            count = 0;
            offset = 0;
            for (int col = columnStartIndex; col > columnStartIndex - 4; col--
)
            {
                if (grid[rowStartIndex + offset, col] == player)
                {
                    count++;
                }
                else
                {
                    count = 0;
                }
                offset++;
                if (count == 4) win = true;
            }
        }
    }

    return win;
}

```

ARTIFICIAL INTELLIGENCE

The public AIMove method is the AI of the program and calculates a random position for the computer's counter to be dropped into. When the client tested the game, he found that he won against the computer every time because the computer would only block a win or make a winning move very rarely because of the random gameplay. The AI is greatly extendible and I could use more complex heuristic algorithmic thinking, using the minimax method, which checks the implications of a move for the next three turns. It uses the possible wins a move creates for the computer and the wins it blocks for the opponent to give each possible move a score and the move with the highest score is then chosen. Using this more complex method would allow the client to play against the computer competitively and develop strategies.

```
public int AIMove()
{
    Random rnd = new Random();
    int pos = rnd.Next(0, 6);

    return pos;
}
```

ENCAPSULATION

Since most global variables are private, the only way to access and change the variables is to use get and set methods. The get methods return the values of the variables, whilst the void set methods change their values. These methods are very simple to code but are essential as they are used throughout the rest of the program to access and change the variables, without calling them directly.

```

public Board getBoard()
{
    return b;
}

public int[,] getGrid()
{
    return grid;
}

public int getCounterP()
{
    return counter;
}

public int getCounter(int x, int y)
{
    return grid[x, y];
}

public Boolean getFullGrid()
{
    Boolean fullGrid = false;
    if (counter > 42) fullGrid = true;
    return fullGrid;
}

public int getWinner()
{
    return w;
}

```

INHERITANCE

Inheritance of classes is an important feature of object-oriented design. As the one player and two player games are both different types of the game, the classes can inherit the shared variables from the super class Game. This stops code from being duplicated, which makes the program more efficient.

```

class One : Game
{
    public One(): base()
    {
    }
}

class Two : Game
{
    public Two(): base()
    {
    }
}

```

POLYMORPHISM

Polymorphism allows inherited classes to change an inherited method from the super class, depending on the function of the child class. The start method is needed in both classes so is written in the Game class as an abstract method. When inherited, both classes override the method and set up the game for the correct number of players. The two player public start method shifts between the option for two different players to drop their counter, whilst the one player method calculates a random move for the computer if it is not the player's turn.

```
public abstract void start(int col);

public override void start(int col)
{
    this.getBoard().addCounter(0, col);

    if (this.getBoard().checkUpDown() == true || this.getBoard().checkDiag()
== true)
    {
        this.getBoard().winner();
        this.getBoard().resetGame();
    }
}

public override void start(int col)
{
    if (this.getBoard().getCounterP() % 2 == 0)
    {
        this.getBoard().addCounter(0, col);
        this.getBoard().addCounter(0, AIMove());
    }

    if (this.getBoard().checkUpDown() == true || this.getBoard().checkDiag()
== true)
    {
        this.getBoard().winner();
        this.getBoard().resetGame();
    }
}
```

(The full source code of the program and evidence of development is in Appendix 2).

CRITERION E: EVALUATION

MEETING THE SUCCESS CRITERIA

After talking to my client, my initial success criteria for my system were to:

- Make use of a graphical user interface to show a graphical representation of the board – *works well as my GUI uses two screens and clearly represents the board. The client said that he liked the colour scheme that was used as a theme and the buttons become visible/invisible depending on if the user needs them.*
- Identify the position where the user wishes to place their counter and replace this position with the counter to “fill” it if the move is valid – *works well in both one player and two player games as a user can add a counter and a try is considered invalid if you attempt to add another counter to full column, although no error is displayed.*
- Correctly identify a situation where a player has won and automatically stop play or identify when the board is full so the game is a draw – *works well as a win in a horizontal, vertical or diagonal direction is found and the correct player is declared the winner. Having completed the testing, the game is declared a draw if the board manages to be filled with no wins.*
- Use algorithmic thinking to identify a situation where the computer AI has won – *works adequately well as the computer's move is automatically chosen and the GUI is updated instantly but the level complexity used is very basic and the AI is extendible.*
- Allow the user to restart a new game – *works as the button to start a new game is available once a game is finished but the client did suggest that it would be better to have this option throughout the game, in case a player wants to forfeit and start a new game.*

(Appendix 3 shows a transcript of the interview with my client after testing the solution).

RECOMMENDATIONS FOR FURTHER IMPROVEMENT

After testing, the feedback that I received from the client and advisor was mainly positive and they suggested some improvements for the solution. The program is very extendible and the main area of concern was the AI as the client complained that he was always winning and could not play against the computer competitively.

Because the AI is random, it is not very intelligent and therefore, easy to beat. This could be improved by using the minmax algorithm that my advisor suggested to add complexity and make the AI more aggressive.

The client suggested the idea of adding a scoreboard on the GUI, which could keep a score of the number of wins that each player has had. This would allow players to play more than one game at a time and the scores could be saved to a secure external text file, where they are available to access by the user later. To further develop the point, my advisor also suggested that the top ten high scores could be calculated by using a simple sorting algorithm. I think this would be a very nice feature to add to the solution to extend it and it's not difficult to code but I was limited in the time I had to finish the solution so could not add this feature.

Also, the client wanted to play with other friends and so wanted to run the game on different computers but he was concerned about security. So, I could add a simple login form, which requires a username and password to open the program, which would make it more secure so only my client could run it.

Overall, the client and advisor were satisfied with the final program and I think I was quite efficient in my coding and managed time well. My choice of Visual Studio C# was a good one as it is multi-platform so my game can be run on any computer with the necessary installed running environment, which the client was very pleased with. It also let me create a satisfactory GUI and allowed me to use encapsulation, inheritance and polymorphism, which made my coding easier to manage and more extensible as object oriented programming can easily be adapted and changed.

APPENDIX 1

Simplified rules of Connect Four:

- 1) Player 1 is chosen to start first.
- 2) Each player drops their counters down one of the slots in the top of the grid.
- 3) The turns alternate until one of the players get four counters of their colour in a row. This can be horizontal, vertical or diagonal.
- 4) The first player to get four in a row wins.
- 5) If the board is filled with counters and neither player gets four in a row, the board is a draw.

APPENDIX 2

Start class

```
public partial class Start : Form
{
    public Start()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        choosePlayer(false);
    }

    private void button2_Click(object sender, EventArgs e)
    {
        choosePlayer(true);
    }

    private void choosePlayer(Boolean twoPlayer)
    {
        Main main = new Main(twoPlayer);
        main.Show();
        this.Visible = false;
    }
}
```

Main class

```
public partial class Main : Form
{
    Game g;
    TextBox[,] boardUI = new TextBox[6,7];
    int players = 0;

    public Main(Boolean twoPlayer)
    {
        InitializeComponent();
        this.Visible = true;

        if (twoPlayer == true)
        {
            g = new Two();
            players = 2;
        }
        else
        {
            g = new One();
            players = 1;
        }

        boardUI[0, 0] = textBox1;
        boardUI[0, 1] = textBox2;
        boardUI[0, 2] = textBox3;
    }
}
```

```

boardUI[0, 3] = textBox4;
boardUI[0, 4] = textBox5;
boardUI[0, 5] = textBox6;
boardUI[0, 6] = textBox7;
boardUI[1, 0] = textBox8;
boardUI[1, 1] = textBox9;
boardUI[1, 2] = textBox10;
boardUI[1, 3] = textBox11;
boardUI[1, 4] = textBox12;
boardUI[1, 5] = textBox13;
boardUI[1, 6] = textBox14;
boardUI[2, 0] = textBox15;
boardUI[2, 1] = textBox16;
boardUI[2, 2] = textBox17;
boardUI[2, 3] = textBox18;
boardUI[2, 4] = textBox19;
boardUI[2, 5] = textBox20;
boardUI[2, 6] = textBox21;
boardUI[3, 0] = textBox22;
boardUI[3, 1] = textBox23;
boardUI[3, 2] = textBox24;
boardUI[3, 3] = textBox25;
boardUI[3, 4] = textBox26;
boardUI[3, 5] = textBox27;
boardUI[3, 6] = textBox28;
boardUI[4, 0] = textBox29;
boardUI[4, 1] = textBox30;
boardUI[4, 2] = textBox31;
boardUI[4, 3] = textBox32;
boardUI[4, 4] = textBox33;
boardUI[4, 5] = textBox34;
boardUI[4, 6] = textBox35;
boardUI[5, 0] = textBox36;
boardUI[5, 1] = textBox37;
boardUI[5, 2] = textBox38;
boardUI[5, 3] = textBox39;
boardUI[5, 4] = textBox40;
boardUI[5, 5] = textBox41;
boardUI[5, 6] = textBox42;
}

private void button1_Click(object sender, EventArgs e)
{
    g.start(0);
    Board b = g.getBoard();
    updateBoardUI(b);
}

private void button2_Click(object sender, EventArgs e)
{
    g.start(1);
    Board b = g.getBoard();
    updateBoardUI(b);
}

private void button3_Click(object sender, EventArgs e)
{
    g.start(2);
    Board b = g.getBoard();
    updateBoardUI(b);
}

```

```

private void button4_Click(object sender, EventArgs e)
{
    g.start(3);
    Board b = g.getBoard();
    updateBoardUI(b);
}

private void button5_Click(object sender, EventArgs e)
{
    g.start(4);
    Board b = g.getBoard();
    updateBoardUI(b);
}

private void button6_Click(object sender, EventArgs e)
{
    g.start(5);
    Board b = g.getBoard();
    updateBoardUI(b);
}

private void button7_Click(object sender, EventArgs e)
{
    g.start(6);
    Board b = g.getBoard();
    updateBoardUI(b);
}

public void updateBoardUI(Board b){
    for (int x = 0; x < 6; x++)
    {
        for (int z = 0; z < 7; z++)
        {
            if(b.getCounter(x,z) == 1){
                boardUI[x,z].BackColor = Color.Maroon;
            }else if(b.getCounter(x,z) == 2){
                boardUI[x,z].BackColor = Color.Goldenrod;
            }
        }
    }

    if (b.getWinner() == 1)
    {
        textBox43.Visible = true;
        textBox43.Text = "PLAYER 1 WINS";
        button8.Visible = true;
    }
    else if (b.getWinner() == 2)
    {
        textBox43.Visible = true;
        if (players == 1)
        {
            textBox43.Text = "COMPUTER WINS";
        }
        else
        {
            textBox43.Text = "PLAYER 2 WINS";
        }
        button8.Visible = true;
    }
}

```

```

    }
    else if (b.getFullGrid() == true)
    {
        textBox43.Visible = true;
        textBox43.Text = "DRAW";
        button8.Visible = true;
    }
}

private void button8_Click(object sender, EventArgs e)
{
    Start start = new Start();
    start.Visible = true;
    this.Visible = false;
}
}

```

Game class

```

abstract class Game
{
    private Board b;

    public Game()
    {
        b = new Board();
    }

    public abstract void start(int col);

    public Boolean addCounter(int column, int counter)
    {
        Boolean addCounter = b.addCounter(column, counter);
        return addCounter;
    }

    public Board getBoard()
    {
        return b;
    }
}

```

Board class

```

public class Board
{
    private int[,] grid = new int[6, 7];
    private int counter = 0;
    private int totalMoves = 0;
    private int w = 0;

    public Board()
    {

```

```

        for (int x = 0; x < 6; x++)
        {
            for (int z = 0; z < 7; z++)
            {
                grid[x, z] = 0;
            }
        }
    }

    public int[,] getGrid()
    {
        return grid;
    }

    public int getCounterP()
    {
        return counter;
    }

    public int getCounter(int x, int y)
    {
        return grid[x, y];
    }

    public Boolean getFullGrid()
    {
        Boolean fullGrid = false;
        if (counter > 42) fullGrid = true;
        return fullGrid;
    }

    public int getWinner()
    {
        return w;
    }

    public Boolean addCounter(int counter, int column)
    {
        Boolean addC = false;
        int x=0;

        if (this.counter % 2 == 0)
        {
            counter = 1;
        }
        else
        {
            counter = 2;
        }

        while (x<6 && grid[x,column] != 0)
        {
            x = x + 1;
        }

        if (x < 6)

```

```

{
    grid[x, column] = counter;
    addC = true;

    this.counter = this.counter + 1;
    this.totalMoves++;
}
else
{
    addC = false;
}
return addC;
}

public Boolean checkUpDown()
{
    int count = 0;
    int player = 0;
    Boolean win = false;

    if (counter % 2 == 0)
    {
        player = 1;
    }
    else
    {
        player = 2;
    }

    for (int row = 0; row < 6; row++)
    {
        for (int col = 0; col < 7 ; col++){

            if (grid[row, col] == player)
            {
                count++;
            }
            else
            {
                count = 0;
            }

            if(count==4) win = true;
        }
    }

    for (int col = 0; col < 7; col++)
    {
        for (int row = 0; row < 6 ; row++){

            if (grid[row, col] == player)
            {
                count++;
            }
            else
            {
                count = 0;
            }

            if(count==4) win = true;
        }
    }
}

```

```

    }

    return win;
}

public Boolean checkDiag()
{
    int count = 0;
    int player = 0;
    Boolean win = false;
    int offset = 0;

    if (counter % 2 == 0)
    {
        player = 1;
    }
    else
    {
        player = 2;
    }

    for (int rowStartIndex = 0; rowStartIndex < 3; rowStartIndex++)
    {
        for (int columnStartIndex = 0; columnStartIndex < 4;
columnStartIndex++)
        {
            count = 0;
            offset = 0;
            for (int col = columnStartIndex; col < columnStartIndex + 4;
col++)
            {
                if (grid[rowStartIndex + offset, col] == player)
                {
                    count++;
                }
                else
                {
                    count = 0;
                }
                offset++;

                if (count == 4) win = true;
            }
        }
    }

    for (int rowStartIndex = 0; rowStartIndex < 3; rowStartIndex++)
    {
        for (int columnStartIndex = 6; columnStartIndex > 2; columnStartIndex-
-)
        {
            count = 0;
            offset = 0;
            for (int col = columnStartIndex; col > columnStartIndex - 4; col--
)
            {
                if (grid[rowStartIndex + offset, col] == player)
                {
                    count++;
                }
                else

```



```

        {
            count = 0;
        }
        offset++;

        if (count == 4) win = true;
    }
}

return win;
}

public void winner()
{
    if (counter%2 == 0)
    {
        w = 1;
    }else if (counter%2 == 1)
    {
        w = 2;
    }
}

public void resetGame()
{
    for (int x = 0; x < 6; x++)
    {
        for (int z = 0; z < 7; z++)
        {
            grid[x, z] = 0;
        }
    }
}

```

OnePlayer class

```

class One : Game
{
    public One(): base()
    {
    }

    public override void start(int col)
    {
        if (this.getBoard().getCounterP() % 2 == 0)
        {
            this.getBoard().addCounter(0, col);
            this.getBoard().addCounter(0, AIMove());
        }

        if (this.getBoard().checkUpDown() == true || this.getBoard().checkDiag()
== true)
        {
            this.getBoard().winner();
            this.getBoard().resetGame();
        }
    }
}

```

```

    public int AIMove()
    {
        Random rnd = new Random();
        int pos = rnd.Next(0, 6);

        return pos;
    }
}

```

TwoPlayer class

```

class Two : Game
{
    public Two(): base()
    {
    }

    public override void start(int col)
    {
        this.getBoard().addCounter(0, col);

        if (this.getBoard().checkUpDown() == true || this.getBoard().checkDiag()
== true)
        {
            this.getBoard().winner();
            this.getBoard().resetGame();
        }
    }
}

```

APPENDIX 3

These questions were asked after the client tested the game

Me: So, generally, what are your thoughts about the final game?

Client: Oh, I like it. I think it looks good and, well, it works...so yeah!

Me: Okay, great. So, let's break it down. What do you think of how it looks? Does it include everything that you wanted?

Client: Yeah. In fact, the screens use my favourite colours, which is cool because it feels like it was made for me. The counters are red and yellow, so it still looks slightly traditional and it's really simple. Oh, also it says 'Player 1 wins' in the middle of the screen and the buttons are only there when I need them, which is cool but I think I would like it if the new game button was always there because sometimes a player may just want to forfeit a game. But, yeah, I don't know, I just think having the option there would be nice.

Me: Yes, that makes sense. And, of course, a very important part of this was the one player game. Having played that now, what do you think?

Client: Well, I don't really know? It's too easy. I played against the computer 5 times and I won each time. The computer didn't even try to win because there were loads of opportunities where the computer could have won but it didn't drop the counter in the right position. So, it's too easy and I can't actually play against it because I will always win. So, the two player is perfect, but the one player definitely needs some work, I think.

Me: Yeah, of course, that's a good point and I noticed it when testing as well and if I had more time, I would have improved the AI. But, moving on, you said you were happy with the two player game?

Client: Yes. It works perfectly. It's just like playing against a normal person in real life but quicker!

Me: Well, that's good to hear! Can you think of any improvements?

Client: Well, I don't know if you can do this but maybe a scoreboard? Like, a board that keeps a score of how many times a player has won so if I play 20 games with someone, we can see how many each one has won. I think that would be cool.

Me: Yeah, that would be a good feature. And maybe, high scores too?

Client: Yeah, like a leader board! And the scores could be saved so I could check them whenever I want.

Me: That sounds like a really good idea! Anything else?

Client: Ummm, nope.

Me: Do you have any questions about the game?

Client: Umm, yeah actually. Can I use the game on another computer like if I went to my friend's house?

Me: Yes. Your friend would need to download a program so that the game can be run but if he has that installed on his computer then you can run the game and save it to his computer.

Client: Okay, and would that mean that he can change the high scores?

Me: Oh, well if the scores are saved, I could add a password to the file so only you can open them. Also, maybe a login form for the game itself? So, the game will open and ask for a username and password to start running. This way, only you can start the game.

Client: Yeah, that's good.

Me: Okay then, I guess that's that finished. Thank you for testing it and I hope you're happy with the final game.

Client: Yeah, it's awesome. Thanks a lot!

APPENDIX 4

Bibliography:

Picture of Connect Four board taken from

<http://www.sci.brooklyn.cuny.edu/~chipp/cis15/hw1/images/a.jpg>
(Accessed 13/10/13).